ECE 3663 - Digital Integrated Circuits
University of Virginia

Design Review II

Report Submitted by:
Team Awesome
Mark Cheung, Michael Recachinas, Piyapath Siratarnsophon, David Yi

On our honor, we have neither given nor received unauthorized assistance on this project.

April 19, 2013

Design Review II

Abstract

The project is to design, simulate, and build an arithmetic logic unit (ALU) contained in a digital signal processor (DSP) using CMOS transistors and Cadence Free PDK design kit. Within the ALU will be basic logic functions (AND, OR, NOT) as well as more complicated functions such as add and subtract. The functions within the ALU are to effectively be chosen by an 8-to-1 multiplexer (MUX). Using latches, ultimately registers will be implemented to store the values on the rising edge of the clock. The goal of the project is to minimize the power dissipation, latency, and overall area of the ALU, while ensuring accurate and precise output.

1 Design Decisions and Justification

1.1 ADD

To implement the 8-bit adder design, we followed the mirror adder design. This design uses minimum size sum gates to reduce the load on carry. It propogates the carry out if inputs A and B are the same i.e. both 0's or both 1's. This decreases the average propagation delay as it does not have to wait for the carry-in half of the time (assuming all inputs are equally likely). The NMOS and PMOS chains are completely symmetrical. Furthermore, there's a maximum of two series transistors in the carry-generate gate. Granted, this design is not as fast as the logarithmic tree adder, but it is faster than the ripple-carry adder.

We used the schematics from the handout given by the TA. This is not the optimal design as we are using an extra inverter for the carry out. What we could have done instead is to use another schematics that invert the A B inputs and alternate it with this one. We decided not to implement this because we plan to use a tree adder in the future that focuses primarily on delay.

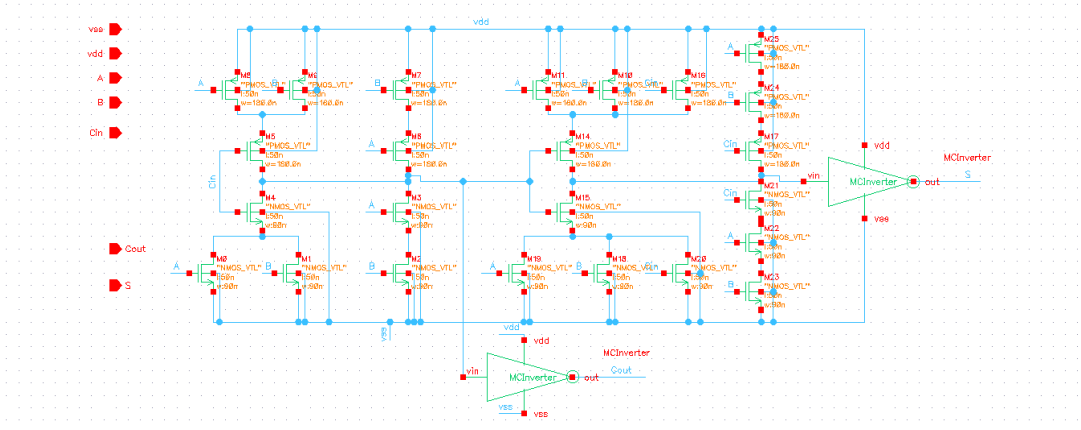

Figure 1: 1-bit Adder Schematic (from *Cadence*)

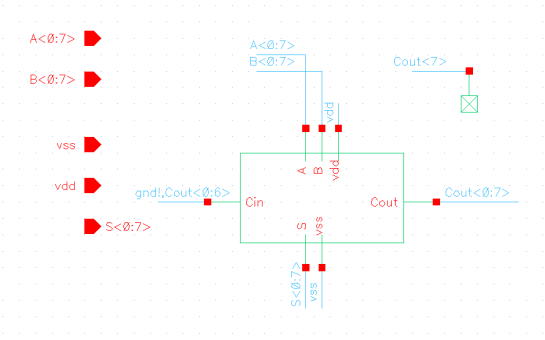

Figure 2: 8-bit Adder Schematic, which is just a cascade of 1-bit adder, with carry out of each adder connected to the carry in of the subsequent one. The first carry-in bit is connected to ground, and the last one connected to a noConn to remove the warnings. (from *Cadence*)

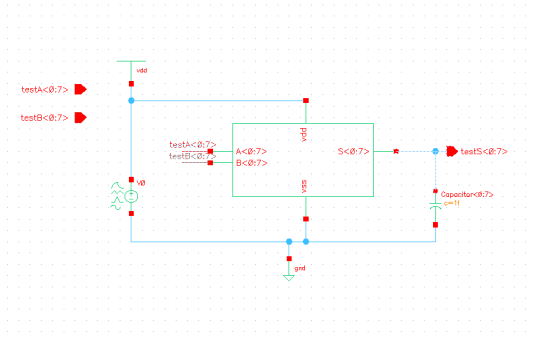

Figure 3: 8-bit Adder Test Bench Schematic with 8 1f capacitors connecting the outputs to ground (for charging and discharging the outputs). (from *Cadence*)

1.2 2COMP

To implement the 8-bit two's complement design, we focused on the delay and area. We used 2:1 MUXs to avoid utilizing an adder that would have otherwise increased the delay. Also, by choosing this design, the area is much smaller than the inverter and adder. As can be seen in figure 4, the least significant bit is passed to the output and the select bit for the 2 MUXs for the next least significant bit. The MUX on the left takes the inverted and the non-invertered data bits. This is selected by the output from the last output and decided whether the current output should be inverted. The previous output also selects whether to pass the output or a bit 1 from the MUX on the right. The output of this MUX is then propagated to the 2 MUXs for the next least significant bit and so forth until bit 8. At bit 8, there is no consecutive data to select and therefore no MUX on the right. On average each bit will have 6.5 CMOS transistors accompanying it, making it 52 CMOS transistors in total for this component.

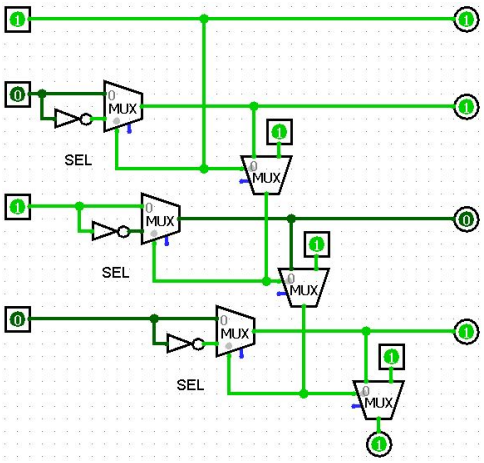

Figure 4: Two's Complement High Level Design (from *Logisim*)

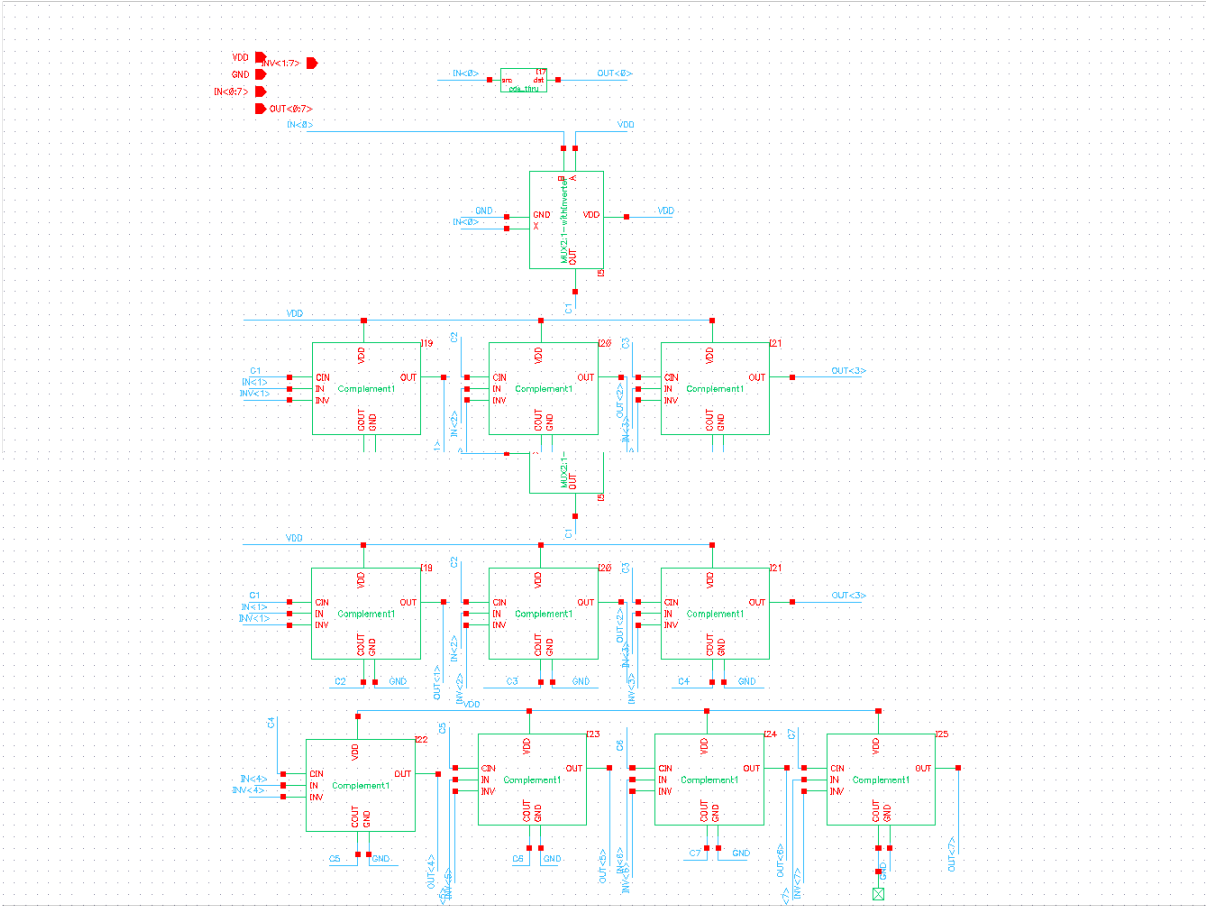

Figure 5: Two's Complement Schematic (from *Cadence*)

1.3 SHIFT

To implement the logical shift, we used just one 8:1 MUX with each input to the MUX selecting portion of input A. For example, bit 0 selects bits 1-7 of the inputs, and the ground and bit 6 selects 2 bits of ground and input 0-5. This design gives a very high area as the 8:1 MUX consists of 7 2:1 MUXes and each 2:1 MUX uses 8 transistors. This leads to a total of 48 2:1 MUXes $\times$ 8 transistors = 384 transistors. This design is also neither the most stable nor the fastest. However, we focused primarily on functionality. For the final delivery, we will most likely follow a barrel-shifter design. The benefit of a barrel-shifter is the ability to shift a word by a specified number of bits in one clock cycle. Also, the number of multiplexers required for an n-bit word is $n \log_2 n$, which in our case would therefore be $8 \log_2 8 = 8 \times 3 = 24$ MUXes (half of what we have here).

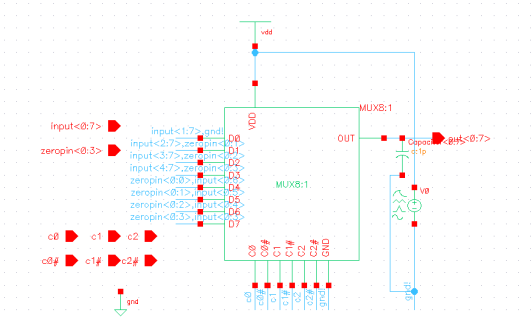

Figure 6: Logical Shift Schematic. For the select bits, bit 0 1 specifies the amount of shifts and bit 2 specifies the direction (from *Cadence*)

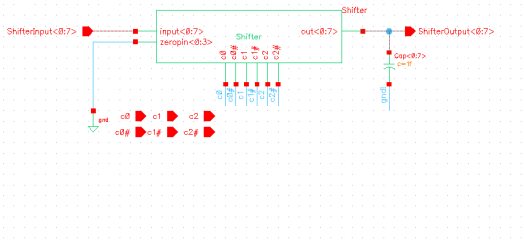

Figure 7: Logical Shift Test Bench Schematic (from *Cadence*)

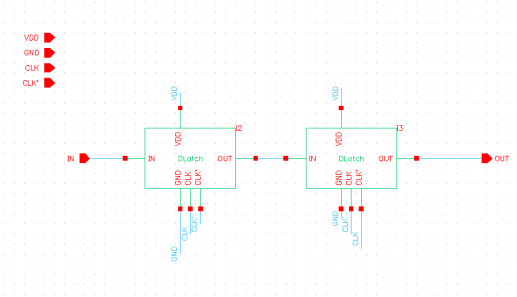

Figure 8: 1-bit Register Schematic (from *Cadence*)

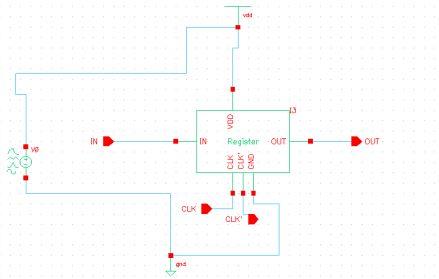

Figure 9: 1-bit Register Test Bench Schematic (from *Cadence*)

1.4 Register

To implement the 8-bit register, we cascaded D-Latches, the typical construction of registers. For D-latch, we have study only one other approach in the recent homework, which uses a total 10 registers (2 extra for the inverter). Hence, we decided to go with the below schematics as it uses 2 less transistors. We intend to test the delay and power (where the other approach might be better) before we settle on this.

From the homework,we studied another schematics for the register. It uses 6 NAND gates, which require more than $6 \times 4 = 24$ transistors due to certain 1 3-input (instead of 2-input) NAND gate. For our register, we coupled two latches together so it uses just $8 \times 2 = 16$ transistors.

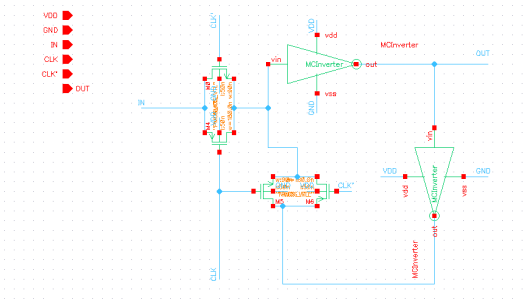

Figure 10: 1-bit D-Latch Schematic (from *Cadence*)

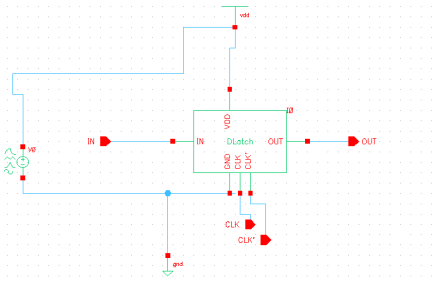

Figure 11: 1-bit D-Latch Test Bench Schematic (from *Cadence*)

1.5 Top Level Connection: Full Schematics

We have all the components needed wired together for the design proejct as shown below. What is left include testing for delays, failure (sweep frequency), hold time, propagation delay, and maximum clock speed (setup time).

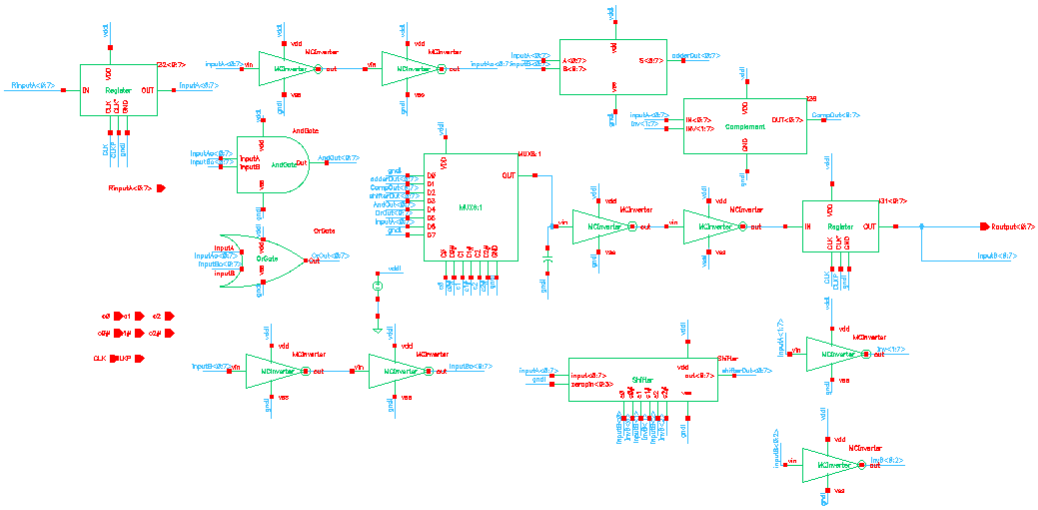

Figure 12: ALU Schematic with all features for Design Review II (from *Cadence*)

2 Simulation Tests and Results

2.1 ADD

We tested the adder by connecting 2 sets of inputs (A B) into the 8b adder schematics shown above (see figure 3: adder test bench). We set A=010X001Y and B=11010111 with X switching between 1 0 at a period of 2 ns and pulse width of 1 ns and Y switching between 1 0 at a period of 1 ns and pulse width of .5 ns. We observe the expected output sum (testS): 00ST10UV. V is the inverse of Y (since it's Y+1), and U is the inverse of V. Bit 3 is 0 because there's always a carry in and it's added to 1 (hence it always has a carry out of 1 as well). Bit 4 is 0+carry in from bit 3. We set both A's B's Bit 4 to 0

to separate testing of different periods. BIT T is the inverse of X (since it's 1+X), BIT S is the inverse of T (since it's X+1 carry-in). BIT 6 is always 0 (1+1) and so is BIT 7 (1+ 1 from carry-in).

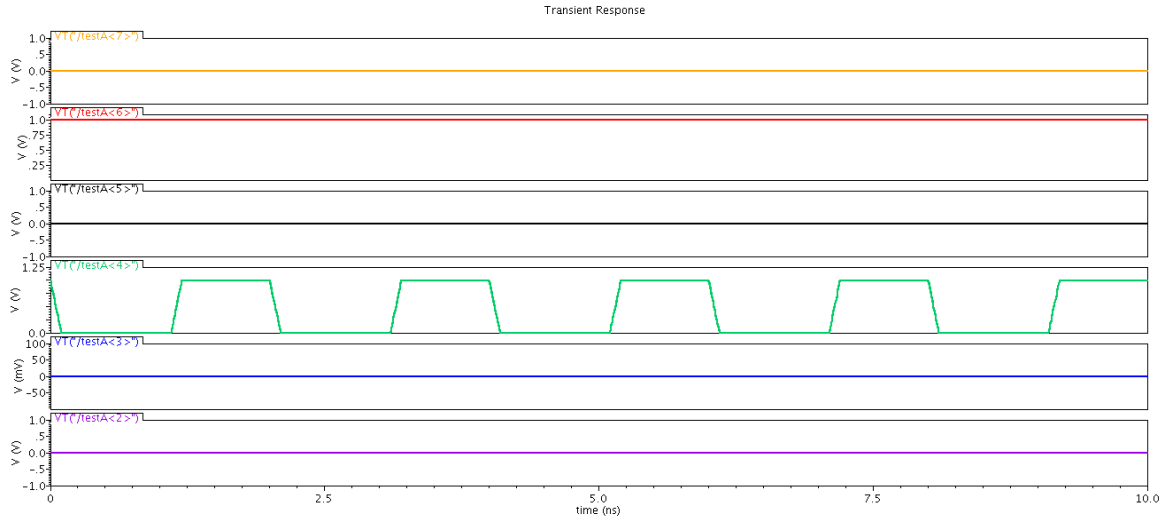

Figure 13: 'A' Input - Adder Transient simulation (from *Cadence*)

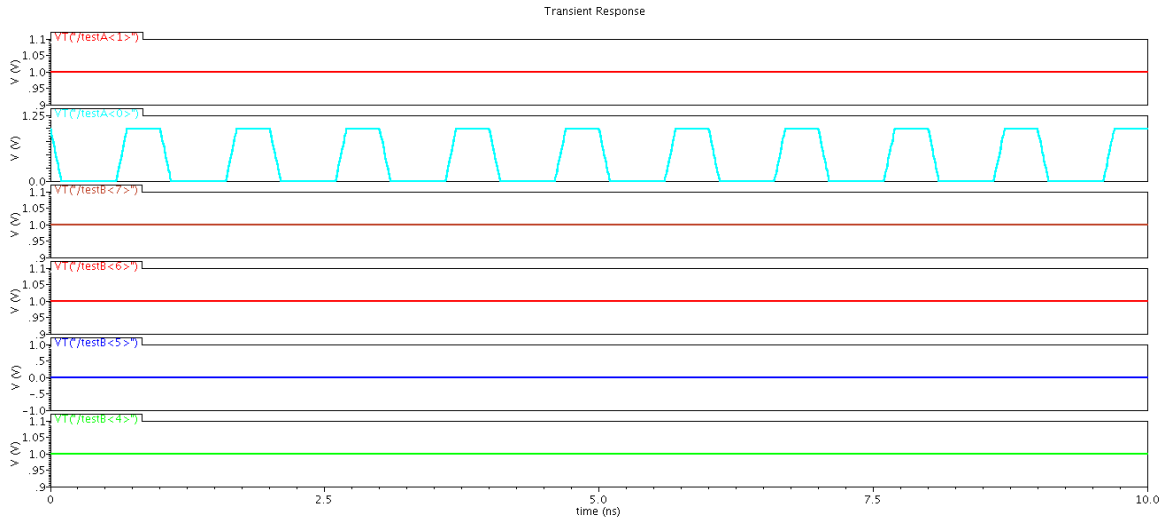

Figure 14: 'A' Input and 'B' Input - Adder Transient simulation (from *Cadence*)

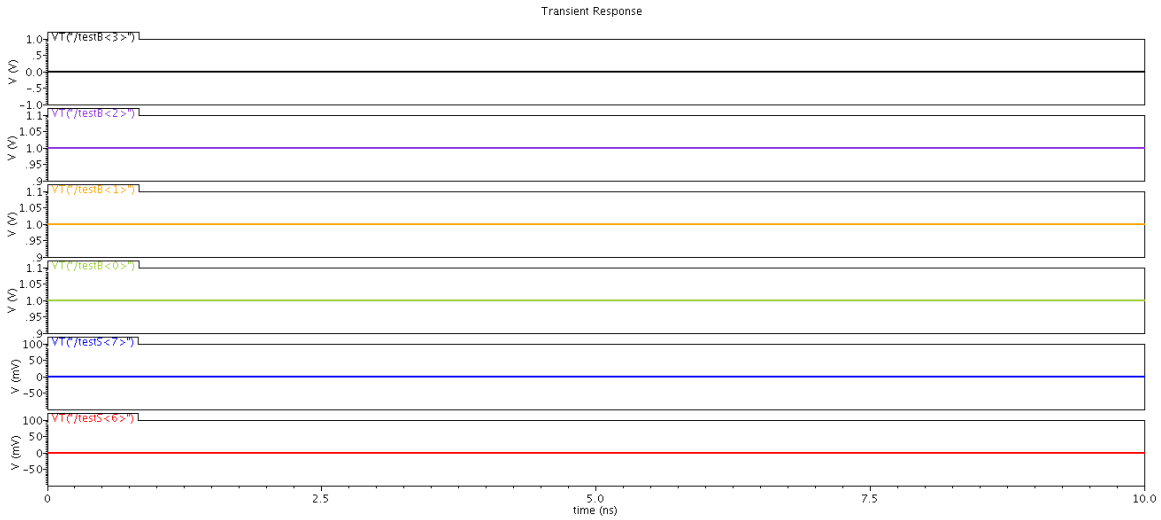

Figure 15: 'B' Input and 'S' Output - Adder Transient simulation (from *Cadence*)

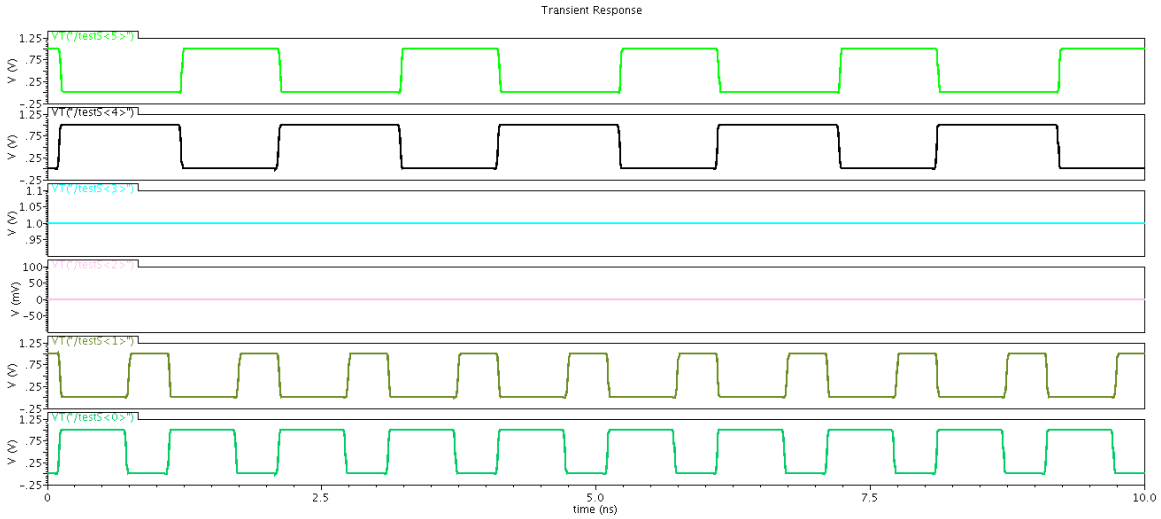

Figure 16: 'S' Ouput - Adder Transient simulation (from *Cadence*)

2.2 2COMP

The 2's complement was tested by setting different inputs and checking if it matches the expected results. For the case shown in the graph below, input is 1001000X where X is a pulse that changes between 0 and 1. The output shown is 01110000 for X equals 0 and 01101111 for X equals 1 which are the expected outputs.

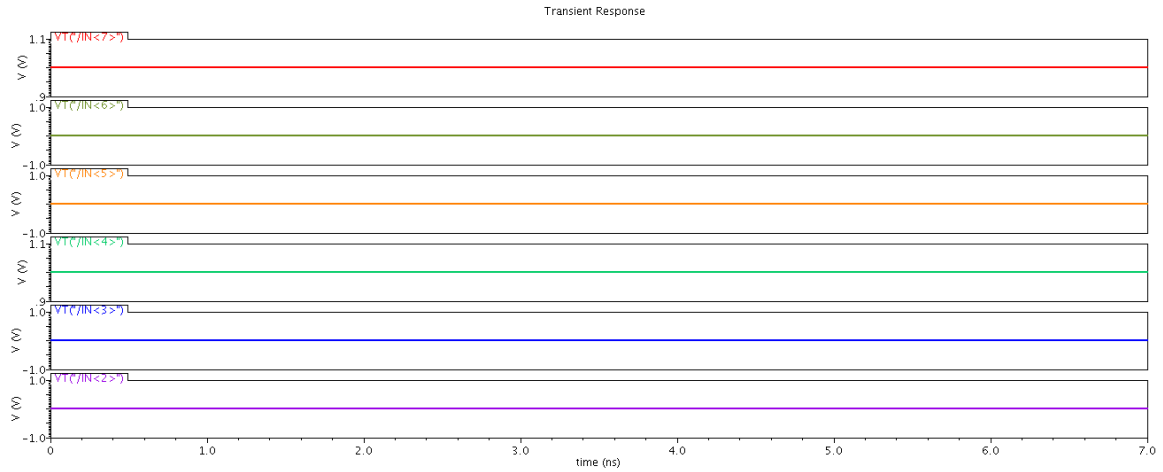

Figure 17: Input - 2COMP Transient simulation (from *Cadence*)

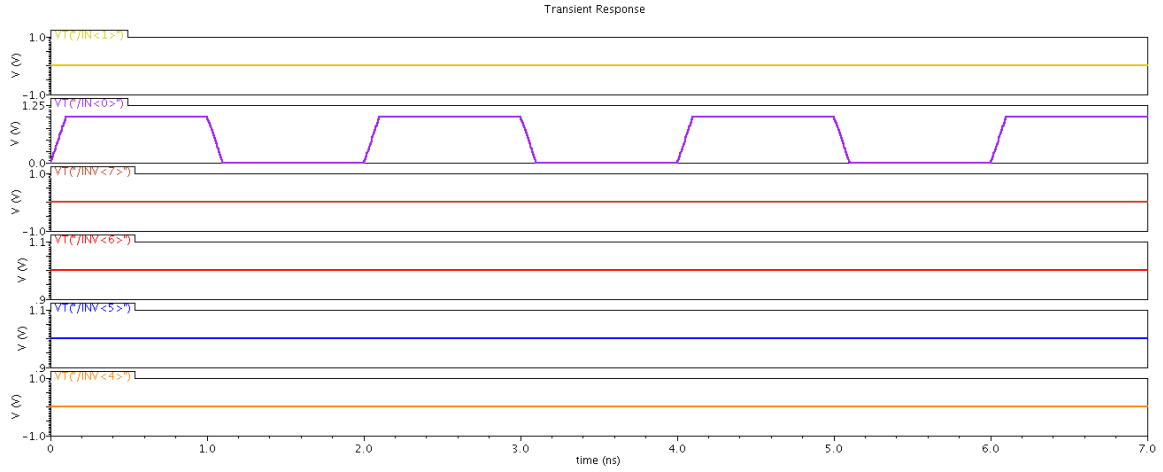

Figure 18: Input and 'Inv' Inverted Input - 2COMP Transient simulation (from *Cadence*)

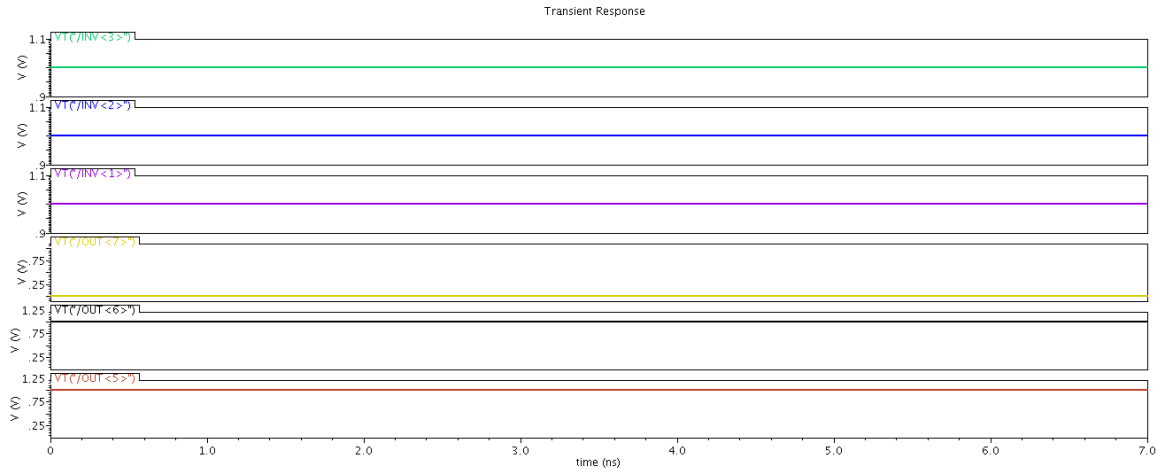

Figure 19: 'Inv' Inverted Input and Output - 2COMP simulation (from *Cadence*)

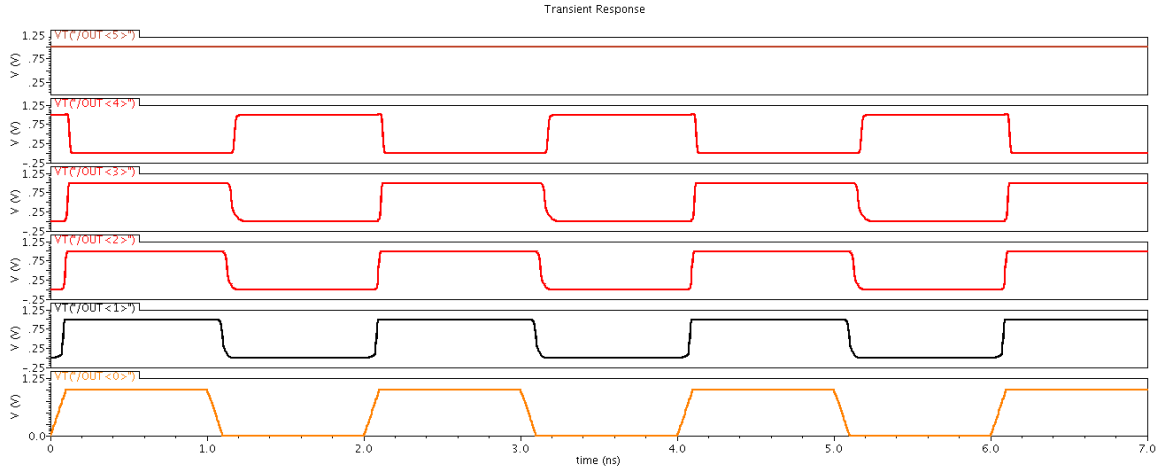

Figure 20: Output - 2COMP simulation (from *Cadence*)

2.3 8-bit Register

To test the register, the D latch was first test by setting the input and output with a period of 2 ns and 3 ns respetively. The following third plot on the figure below shows the output which is inverted from the actual component. When the CLK is low, the D latch is in transparent mode and the output is propagated from the input. When the CLK is high, the latch is in hold mode.

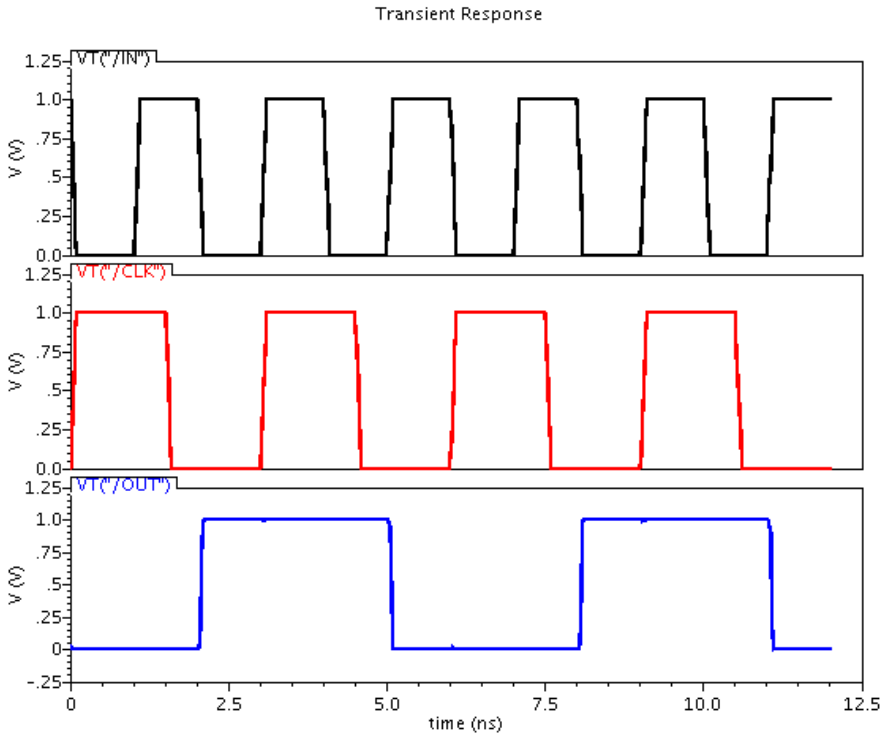

Figure 21: Input, CLK, and Output, respectively - 1-bit D-Latch simulation (from *Cadence*)

The next graph and the one after shows the test case for the actual register. For figure 22, the register only read at the rising edge of the CLK. The register will store the value for one CLK period when it is high.

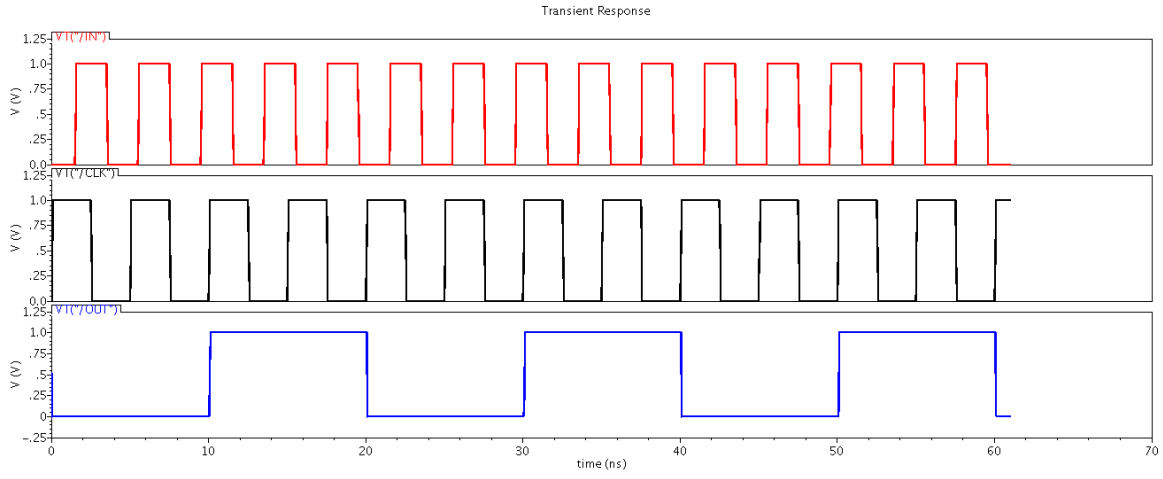

Figure 22: Input, CLK, and Output, respectively - 1-bit Register simulation (from *Cadence*)

The following takes a closer look with less periods but greater difference between the CLK and input periods. As expected, the value of the output is high when the CLK is high and it continues propagating D to the output for each CLK period.

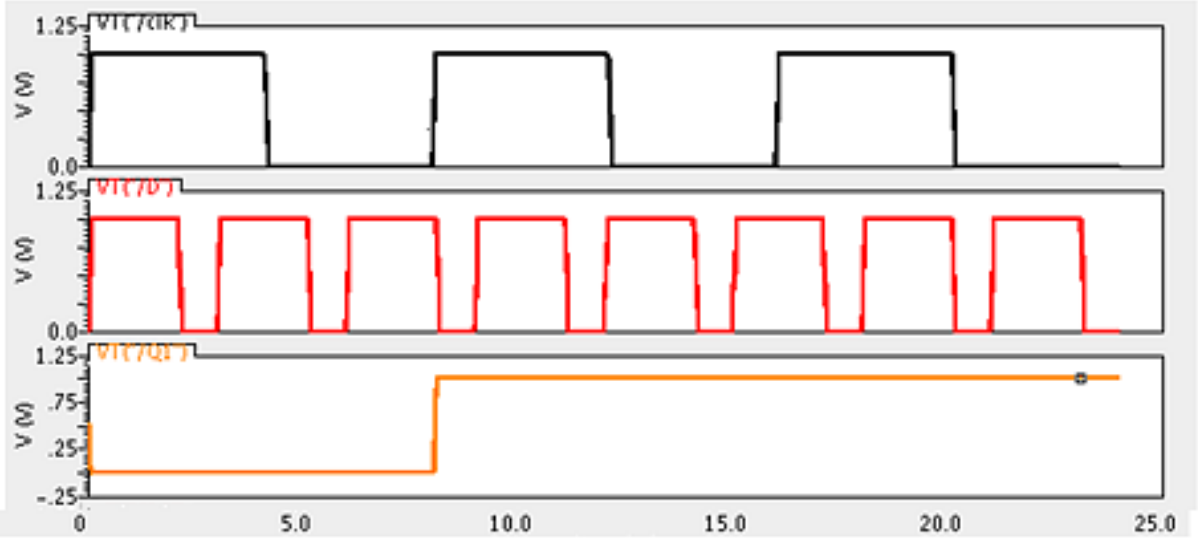

Figure 23: Input, CLK, and Output, respectively - 1-bit Register simulation (from *Cadence*)

2.4 SHIFT

We tested the shifter with the select inputs of 011, which corresponds a right shift of 4. For the data inputs, we specify increasing period and pulse width starting from bit 0 of the input (period of 2ns and pulse width of 1ns). The results show that bit 4-7 of the inputs are shifted to bits 0-3 and replaced by logical 0's.

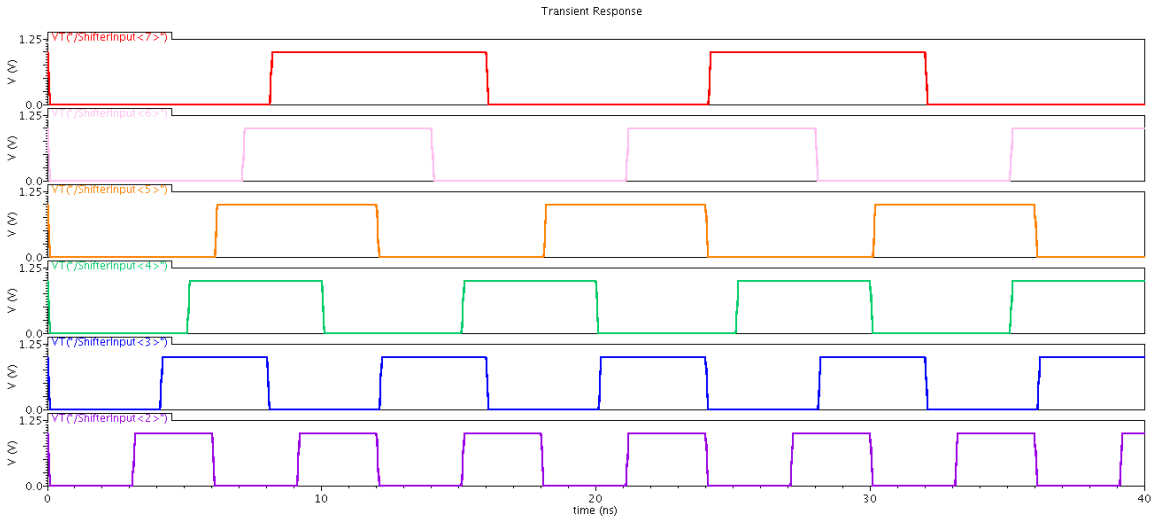

Figure 24: Input - SHIFT simulation (from *Cadence*)

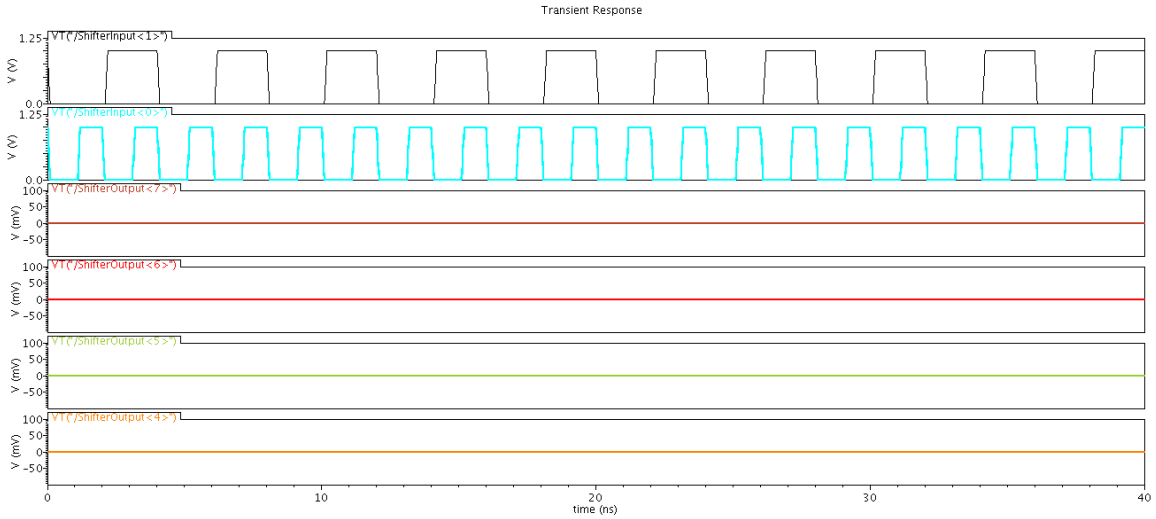

Figure 25: Input and Output - SHIFT simulation (from *Cadence*)

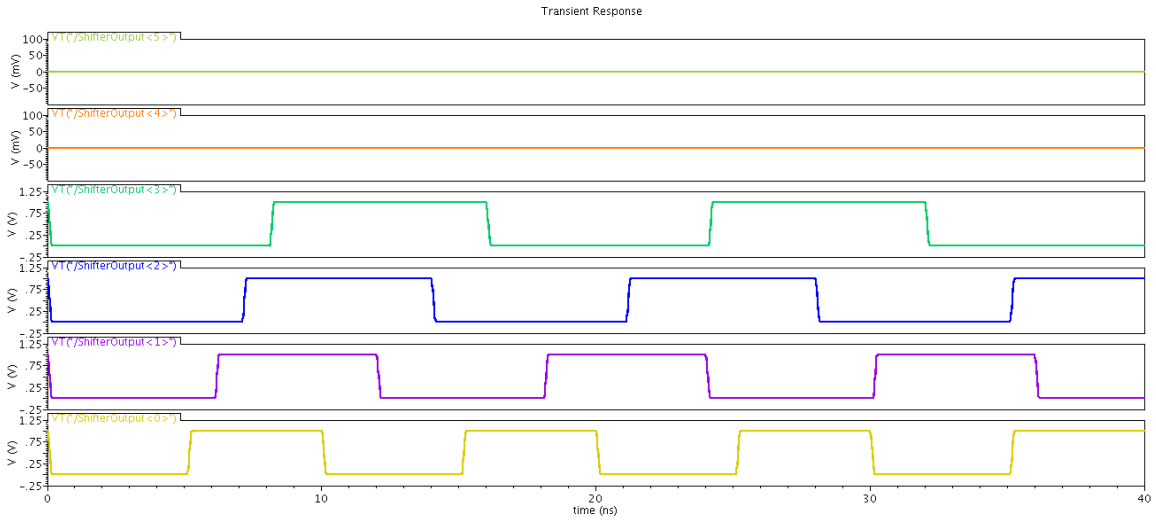

Figure 26: Output - SHIFT simulation (from *Cadence*)

3 Delay

Table 1: Table of each component's worst case delay. The worst case delays were calculated using the Cadence calculator. We tested each delay individually and added 38.59 for the 8:1 MUX delay (for selecting the operation)

Operation	Worst-Case Delay	Scenario
ADD	265.1 + 38.59 ps = 303.69 ps	When $A \bigoplus B = 1$, the adder must wait for C_{in} to propagate to C_{out}. In the stimulli, we set it up so that A is all 1's and B is all 0's except for the first bit (pulse input). We measure the delay between the rising edge of the first bit of B and the last sum bit.
2COMP	112.2 + 38.59 ps = 150.79 ps	When input is all 0, the control signals have to propagate from bit 0 to bit 8. This is because all bits have to transition from 0 to 1. The component gives output based on previous most significant bit and therefore propagates the output when all bits transition.
SHIFT	38.61 + 38.59 ps = 77.22 ps	The delay is the same as the pass because we use the 8:1 MUX for both the shift and the ALU. The delay comes from charging/discharging the capacitors when the input switch from 0 to 1 or vice versa. We measure the delay by first setting the input A to a pulse function and measure the propagation between its rising edge/falling edge to the rising edge/falling edge of the corresponding shifted result. Both edges yield values of around 38.6 ps.
AND	24.4 + 38.59 ps = 62.99 ps	When both inputs change from 0 to 1. The delay will then be obtained from the NMOS in series.
OR	19.79 + 38.59 ps = 58.38 ps	When both inputs change from 1 to 0. The delay will come from the discharging of the PMOS in series.
Pass A	38.59 ps	The delay comes from the MUX as no other component is used for the Pass. (same as shift). The delay comes from charging/discharging the capacitors when the input switch from 0 to 1 or vice versa. We measure the delay by first setting the input A to a pulse function and measure the propagation between its rising edge/falling edge to the rising edge/falling edge of the corresponding shifted result. Both edges yield values of around 38.6 ps.

4 Design/Test Strategy for Final Review

Since we are not asked to take into account delay, power, and area for this design review, we simply commented on the desing decisions we made here. For the final review, we plan to test the built components by simulating each input with injecting direct current (DC) input as well as pulses.

Our first priority is delay, then area and power as the benchmark is $\text{delay}^2 * power * area$.

4.1 Component Testing

Up to this design review, we have tested each component and make sure that they are working correctly. For the final review, we intend to setup automatic testing in ocean for the full schematics (Figure 12).

4.2 Performance Improvement

4.2.1 Setup Time

We plan to do a sweep frequency to find the setup times of each component. We will measure the setup delays by finding the minimum amount of time the inputs must be stable before switching (clock/select bits). This is directly related to the maximum clock frequency we can use.

4.2.2 Propagation Delay

In this design review, we tested for the worst-case delay for the propagation (see table above). Our table shows that the worst-case delay lies with the adder, so our top priority for the next design review is to use the logarithmic tree adder (O(log n) time, where n is the number of data inputs, which has a better performance (lower delay, but at a trade-off of higher power and area). Two of our group members did a lot of research on it.

4.3 Power Management

For the final design review, we intend to minimize the power by turning off the components that are not used. All the subcircuits of the ALU run regardless of whether they are being used, and in doing so consume unnecessary power. We plan to add some kind of enabler to shut down the unused subcircuit components.

4.4 Area Optimization

To find the area, we simply have to add up the sum of transistors (it's W/L). We will keep the area in mind as we look for other components (especially adder) to optimize the delay.

5 Arbitrary Function

After meeting with Professor Blalock, we have decided to build an 8 bit comparator. The comparator takes two numbers as input in binary form and determines whether one number is greater than, less than, or equal to the other number. The component is very useful for many higher level operations e.g. branching. This component was chosen over the multiplier because our ultimate goal is minimizing delay, and the multiplier delay will be much greater than the comparator delay.

6 Work Breakdown Schedule

6.1 Current Group Progress

Currently, the team has successfully made the 8-bit ADD, 8-bit 2COMP, SHIFT, 8-bit Register, and top level connections. All of the schematics have been created in Cadence and then tested with Ocean. The purpose of this review to have all of the 8 components working, connected, and tested with the worst case delay taken into account. Schematic design, test documentation, plan, and logged meetings can be found on our Wiki:

`https://venividiwiki.ee.virginia.edu/mediawiki/index.php/ClassECE3663Spring13/ClassECE3663Spring13ProjectAwesome`

6.2 Plan for Design Review II

From the Work Breakdown Schedule for Design Review I, all were successfully completed by April 18 (given the extension of the deadline).

- ☑ April 6, 2013 – ADD, 2 COMP, XOR and SHIFT schematics should be done by then. On Saturday 6, the group will meet in Rice 414 to test the components and perform debugging together.
- ☑ April 7, 2013 – All components should be tested by Sunday, but if they are not, the group will finish debugging the functions by this date. The components will be put together into one ALU by the whole group. A design review 2 report will be put together at the end. When the above is done, the group will make a presentation and decide on roles.
- ☑ April 9, 2013 – The members will meet together to rehearse and review their presentation. By Tuesday, the DSP system should be fully functional with enough tests completed successfully to verify proper functionality.

6.3 Plan for Final Delivery

- → April 25 – Complete the Cadence schematic for the 8-bit comparator (arbitrary function)
  - Michael Recachinas
  - David Yi
- → April 26 – Input power efficiency optimized
  - Mark Cheung
  - Piyapath Siratarnsophon
- → April 26 – Most of the components must be optimized and/or finalized (for delivery)
  - Everyone
- → April 27 – All components must be finalized

- Everyone

→ April 28 – Testing must be complete

- Everyone

→ April 29 – Powerpoint and final report must be complete.

- Everyone

7 Appendix

1. Sample Ocean Code for testing the shifter

```
;; DEFINE OUTPUT PRINT FILE
of = outfile( "output.txt" "w" )

;; SET SIMULATOR
simulator( 'spectre )

;; SET DESIGN
design( "netlist" )


;; SET parameters as needed
;; any parameters defined in the netlist will keep those values -
;; do not redefine here
desVar( "pvdd" 1.1 )
pvdd=1.1

;; SET TEMPERATURE
temp( 25 )

;; SET RESULT DIRECTORY
sprintf(dir "./TranResults")
resultsDir( dir )

 ;; RUN ANALYSIS
 analysis( 'tran ?start 0 ?stop 80n ?step .1n ?strobeperiod .1n
     ?errpreset 'conservative )
 save( 'all )
 run()

 ;; PLOT THE SIGNALS
 selectResults('tran)
plot(getData("ShifterInput\\<0\\>") getData("ShifterInput\\<1\\>")
     getData("ShifterInput\\<2\\>") getData("ShifterInput\\<3\\>")
     getData("ShifterInput\\<4\\>") getData("ShifterInput\\<5\\>")
     getData("ShifterInput\\<6\\>") getData("ShifterInput\\<7\\>")  )
plot(getData("ShifterOutput\\<0\\>") getData("ShifterOutput\\<1\\>")
     getData("ShifterOutput\\<2\\>") getData("ShifterOutput\\<3\\>")
     getData("ShifterOutput\\<4\\>") getData("ShifterOutput\\<5\\>")
     getData("ShifterOutput\\<6\\>") getData("ShifterOutput\\<7\\>"))
    ;; Set variables to be used in simulation (load_pmos and load_nmos are netlis
desVar( "vdd" 1.1 )
desVar( "load_nmos" "90n*mult_fact" )
desVar( "load_pmos" "180n*mult_fact" )
vdd=1.1
mult_fact = FO4
```

```
load_nmos = 90n * mult_fact
load_pmos = 180n * mult_fact

option( 'dochecklimit  "no" )

analysis('tran ?stop "300n"  ?errpreset "moderate"  )
temp( 27 )
run()

selectResult( 'tran )

diff = abs((cross(getData("ShifterOutput\\<6\\>") vdd/2 1 "either" nil nil) - cro
     fprintf( of_csv "%d", diff)


close( of )
```

2. Netlist result of Shifter Test

```
// Generated for: spectre
// Generated on: Apr 14 16:05:20 2013
// Design library name: DIC
// Design cell name: testShifter
// Design view name: schematic
simulator lang=spectre
global 0 vdd!
include "/net/plato.ee.virginia.edu/app/lib/freepdk45/trunk/ncsu_basekit/
     models/hspice/tran_models/models_nom/NMOS_VTL.inc"
include "/net/plato.ee.virginia.edu/app/lib/freepdk45/trunk/ncsu_basekit/
     models/hspice/tran_models/models_nom/PMOS_VTL.inc"

// Library name: DIC
// Cell name: MUX2:1
// View name: schematic
subckt _sub0 A B GND OUT VDD X X\#
    M2 (OUT X\# B GND) NMOS_VTL w=90n l=50n as=9.45e-15 ad=9.45e-15 \
        ps=300n pd=300n ld=105n ls=105n m=1
    M0 (OUT X A GND) NMOS_VTL w=90n l=50n as=9.45e-15 ad=9.45e-15 ps=300n \
        pd=300n ld=105n ls=105n m=1
    M3 (OUT X B VDD) PMOS_VTL w=180.0n l=50n as=1.89e-14 ad=1.89e-14 \
        ps=390.0n pd=390.0n ld=105n ls=105n m=1
    M1 (OUT X\# A VDD) PMOS_VTL w=180.0n l=50n as=1.89e-14 ad=1.89e-14 \
        ps=390.0n pd=390.0n ld=105n ls=105n m=1
ends _sub0
// End of subcircuit definition.

// Library name: DIC
```

```
// Cell name: MUX8:1
// View name: schematic
subckt _sub1 C0 C0\# C1 C1\# C2 C2\# D0 D1 D2 D3 D4 D5 D6 D7 GND OUT VDD
    I7 (D6 D7 GND net38 VDD C0\# C0) _sub0
    I9 (D2 D3 GND net52 VDD C0\# C0) _sub0
    I8 (D4 D5 GND net45 VDD C0\# C0) _sub0
    I10 (D0 D1 GND net59 VDD C0\# C0) _sub0
    I11 (net59 net52 GND net31 VDD C1\# C1) _sub0
    I12 (net45 net38 GND net24 VDD C1\# C1) _sub0
    I13 (net31 net24 GND OUT VDD C2\# C2) _sub0
ends _sub1
// End of subcircuit definition.

// Library name: DIC
// Cell name: Shifter
// View name: schematic
subckt Shifter c0 c0\# c1 c1\# c2 c2\# input\<0\> input\<1\> input\<2\> \
        input\<3\> input\<4\> input\<5\> input\<6\> input\<7\> out\<0\> \
        out\<1\> out\<2\> out\<3\> out\<4\> out\<5\> out\<6\> out\<7\> \
        zeropin\<0\> zeropin\<1\> zeropin\<2\> zeropin\<3\>
    MUX8\:1\<0\> (c0 c0\# c1 c1\# c2 c2\# input\<1\> input\<2\> input\<3\> \
        input\<4\> zeropin\<0\> zeropin\<0\> zeropin\<0\> zeropin\<0\> 0 \
        out\<0\> vdd!) _sub1
    MUX8\:1\<1\> (c0 c0\# c1 c1\# c2 c2\# input\<2\> input\<3\> input\<4\> \
        input\<5\> input\<0\> zeropin\<1\> zeropin\<1\> zeropin\<1\> 0 \
        out\<1\> vdd!) _sub1
    MUX8\:1\<2\> (c0 c0\# c1 c1\# c2 c2\# input\<3\> input\<4\> input\<5\> \
        input\<6\> input\<1\> input\<0\> zeropin\<2\> zeropin\<2\> 0 \
        out\<2\> vdd!) _sub1
    MUX8\:1\<3\> (c0 c0\# c1 c1\# c2 c2\# input\<4\> input\<5\> input\<6\> \
        input\<7\> input\<2\> input\<1\> input\<0\> zeropin\<3\> 0 \
        out\<3\> vdd!) _sub1
    MUX8\:1\<4\> (c0 c0\# c1 c1\# c2 c2\# input\<5\> input\<6\> input\<7\> \
        zeropin\<0\> input\<3\> input\<2\> input\<1\> input\<0\> 0 \
        out\<4\> vdd!) _sub1
    MUX8\:1\<5\> (c0 c0\# c1 c1\# c2 c2\# input\<6\> input\<7\> \
        zeropin\<0\> zeropin\<1\> input\<4\> input\<3\> input\<2\> \
        input\<1\> 0 out\<5\> vdd!) _sub1
    MUX8\:1\<6\> (c0 c0\# c1 c1\# c2 c2\# input\<7\> zeropin\<0\> \
        zeropin\<1\> zeropin\<2\> input\<5\> input\<4\> input\<3\> \
        input\<2\> 0 out\<6\> vdd!) _sub1
    MUX8\:1\<7\> (c0 c0\# c1 c1\# c2 c2\# 0 zeropin\<1\> zeropin\<2\> \
        zeropin\<3\> input\<6\> input\<5\> input\<4\> input\<3\> 0 \
        out\<7\> vdd!) _sub1

    V0 (vdd! 0) vsource dc=1 type=dc
    Capacitor\<0\> (out\<0\> 0) capacitor c=1f
    Capacitor\<1\> (out\<1\> 0) capacitor c=1f
```

```
    Capacitor\<2\> (out\<2\> 0) capacitor c=1f
    Capacitor\<3\> (out\<3\> 0) capacitor c=1f
    Capacitor\<4\> (out\<4\> 0) capacitor c=1f
    Capacitor\<5\> (out\<5\> 0) capacitor c=1f
    Capacitor\<6\> (out\<6\> 0) capacitor c=1f
    Capacitor\<7\> (out\<7\> 0) capacitor c=1f
ends Shifter
// End of subcircuit definition.

// Library name: DIC
// Cell name: testShifter
// View name: schematic
I0 (c0 c0\# c1 c1\# c2 c2\# ShifterInput\<0\> ShifterInput\<1\> \
        ShifterInput\<2\> ShifterInput\<3\> ShifterInput\<4\> \
        ShifterInput\<5\> ShifterInput\<6\> ShifterInput\<7\> \
        ShifterOutput\<0\> ShifterOutput\<1\> ShifterOutput\<2\> \
        ShifterOutput\<3\> ShifterOutput\<4\> ShifterOutput\<5\> \
        ShifterOutput\<6\> ShifterOutput\<7\> 0 0 0 0) Shifter

VCIN0 ( co 0 ) vsource type=dc dc=1;
VCIN0P ( co\# 0 ) vsource type=dc dc=0;
VCIN1 ( c1 0 ) vsource type=dc dc=1;
VCIN1P ( c1\# 0 ) vsource type=dc dc=0;
VCIN2 ( c2 0 ) vsource type=dc dc=0;
VCIN2P ( c2\# 0 ) vsource type=dc dc=1;


VSIN0 ( ShifterInput\<0\> 0 ) vsource type=pulse val0=0
     val1=pvdd delay=0 rise=0.01n fall=0.01n width=1n period=2n
VSIN1 ( ShifterInput\<1\> 0 ) vsource type=pulse val0=0 v
     al1=pvdd delay=0 rise=0.01n fall=0.01n width=2n period=4n
VSIN2 ( ShifterInput\<2\> 0 ) vsource type=pulse val0=0
     val1=pvdd delay=0 rise=0.01n fall=0.01n width=3n period=6n
VSIN3 ( ShifterInput\<3\> 0 ) vsource type=pulse val0=0
     val1=pvdd delay=0 rise=0.01n fall=0.01n width=4n period=8n
VSIN4 ( ShifterInput\<4\> 0 ) vsource type=pulse val0=0
     val1=pvdd delay=0 rise=0.01n fall=0.01n width=5n period=10n
VSIN5 ( ShifterInput\<5\> 0 ) vsource type=pulse val0=0
     val1=pvdd delay=0 rise=0.01n fall=0.01n width=6n period=12n
VSIN6 ( ShifterInput\<6\> 0 ) vsource type=pulse val0=0
     val1=pvdd delay=0 rise=0.01n fall=0.01n width=7n period=14n
VSIN7 ( ShifterInput\<7\> 0 ) vsource type=pulse val0=0
     val1=pvdd delay=0 rise=0.01n fall=0.01n width=8n period=16n
```